Patching task-level robot controllers based on a local μ-calculus formula

Scott C. Livingston Pavithra Prabhakar Alex B. Jose Richard M. Murray

***Abstract*— We present a method for mending strategies for GR(1) specifications. Given the addition or removal of edges from the game graph describing a problem (essentially transition rules in a GR(1) specification), we apply a μ-calculus formula to a neighborhood of states to obtain a "local strategy" that navigates around the invalidated parts of an original synthesized strategy. Our method may thus avoid global resynthesis while recovering correctness with respect to the new specification. We illustrate the results both in simulation and on physical hardware for a planar robot surveillance task.**

I. Introduction

A major research theme of the last two decades is to bring verification and synthesis methods to bear on problems in hybrid systems. For example, see Davoren and Nerode [1] and other papers in that special issue of *Proceedings of the IEEE*; an early example of theorem-proving based verification for control problems is [2]. However, one quickly arrives at undecidable problems (e.g., see [3], [4] and references therein). Thus a current topic is to find interesting classes of hybrid control systems that admit tractable synthesis algorithms—at least in appropriate cases. A motivation for the present work is to exploit additional structure not available in the classical automata-theoretic formulation of reactive synthesis.

Perhaps the simplest structure provided by hybrid systems is a notion of "localness." Given a vertex in a discrete transition system, how can we find other vertices that are sufficiently similar? If the set of vertices maps into a metric space, then immediately we can look at a ball centered on the given vertex to form a "neighborhood of vertices." This is just a basic question of how to cluster data and arises in many contexts. Similarly, a metric may be provided directly for the nodes of an automaton, e.g., as used to formulate robustness in [5].

In the present work, we are concerned with using localness to estimate what parts of a control strategy must be updated in response to a game change. In particular we consider application of these methods in mobile planar robotics. A typical problem in this setting is updating a map describing the locations of obstacles and other areas of interest, such as a charging station. One representation often used is cell decompositions [6], which may be polygonal partitions of part of the state space (such as position but not velocity). If appropriate properties relating the continuous dynamics to discrete movement among cells are satisfied [7] (essentially so that a bisimulation is obtained), then it suffices to solve the synthesis problem over this discrete abstraction. Forming discrete abstractions in this manner and then synthesizing game strategies on the resulting game graph forms the basis of some previous work in robotics [8], [9]. While this situation is superficially very attractive, the number of cells required to solve a problem may be large, and when combined with uncontrolled environment behavior, the cardinality of the discrete abstraction may be such that we have gained little from the abstraction process.

Given that a global strategy may be hard to construct and fragile in settings with inherent uncertainty, one may ask whether local parts of the strategy can be updated as new data is accumulated online. That is, can we perform incremental synthesis by working with neighborhoods of nodes based on metrics from the underlying continuous space?

In [10], Livingston *et al.* address a similar problem as in the present work. However, they treat the synthesis algorithm as a "black box" to be invoked with local specifications, which are then used to mend the original strategy. Here we explicitly propose a μ-calculus formula with which to synthesize a local strategy.

http://resolver.caltech.edu/CaltechCDSTR:2012.003

S.C. Livingston, A.B. Jose, and R.M. Murray are with the California Institute of Technology, Pasadena, CA, slivingston@caltech.edu, ajose@caltech.edu, murray@cds.caltech.edu. P. Prabhakar is with IMDEA Software Institute, Madrid, Spain, pavithra.prabhakar@imdea.org

II. Synthesis Problem

A. GR(1) specifications and associated game

For convenience, several LTL operators are listed in Table I. For an introduction to relevant theory see, e.g., [11]. Let $\mathcal{X}$ be a set of uncontrolled variables, and $\mathcal{Y}$ a set of controlled variables. Each variable has a finite domain, and the product of these domains gives the discrete state set Γ. Restrictions to domains of uncontrolled and controlled variables are denoted by $\Gamma_{\mathcal{X}}$ and $\Gamma_{\mathcal{Y}}$, respectively. Denote the projection of a state $s \in \Gamma$ onto the variables in $\mathcal{X}$ by $s \Downarrow_{\mathcal{X}}$. For a set $B \subseteq \Gamma$, projection is elementwise and denoted by $B \Downarrow_{\mathcal{X}}$. Hence to be precise, $\Gamma_{\mathcal{X}} := \Gamma \Downarrow_{\mathcal{X}}$ and $\Gamma_{\mathcal{Y}} := \Gamma \Downarrow_{\mathcal{Y}}$. A state taken directly from a restricted domain will be marked with an appropriate subscript, e.g., $s_{\mathcal{X}} \in \Gamma_{\mathcal{X}}$. Compositions of these to form a complete state will be written as tuples, e.g., $(s_{\mathcal{X}}, s_{\mathcal{Y}}) \in \Gamma$. As in the literature, we also refer to $\mathcal{X}$ as "environment variables" and $\mathcal{Y}$ as "system variables."

As defined in [12], a GR(1) (Generalized Reactivity of rank 1) specification is of the form

$$\theta_e \wedge \Box \rho_e \wedge \left(\bigwedge_{j=0}^{m-1} \Box \diamondsuit J_j^e \right) \implies \theta_s \wedge \Box \rho_s \wedge \left(\bigwedge_{i=0}^{n-1} \Box \diamondsuit J_i^s \right) \tag{1}$$

where θ_e, θ_s, $J_0^e, \ldots, J_{m-1}^e$, and $J_0^s, \ldots, J_{n-1}^s$ are state formulae on $\mathcal{X} \cup \mathcal{Y}$, ρ_e is a state formula on $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X}$, and ρ_s is a state formula on $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X} \cup \bigcirc \mathcal{Y}$. Application of

TABLE I
SEVERAL LINEAR TEMPORAL LOGIC (LTL) OPERATORS

$\bigcirc$	"next"
$\Diamond$	"eventually"
$\Box$	"always" (safety)
$\Box\Diamond$	"infinitely often" (progress)
$\models$	"satisfies"

the "next" operator $\bigcirc$ to a variable denotes the value taken by that variable at the next time step. E.g., $x \to \bigcirc y$ is a transition rule requiring that from any state where x holds, y must hold at the next time step.

The problem setting is best viewed as a game between the environment, who sets variables in $\mathcal{X}$, and the system, who sets variables in $\mathcal{Y}$. (See the definition of simulation game structure in [12].) The transition rules ρ_e and ρ_s dictate how the environment may move from a particular state, and how the system may then respond. The allowed moves can be expressed in a graph like structure called the *game graph*.

Given a GR(1) formula φ as in equation (1), a *game graph* corresponding to it, denoted G_φ is given by $(\Gamma, \mathrm{E_e}, \mathrm{E_s})$, where

- $\mathrm{E_e}(s) := \{a_\mathcal{X} \in \Gamma_\mathcal{X} \mid (s, a_\mathcal{X}) \models \rho_e\}$,
- $\mathrm{E_s}(s, a_\mathcal{X}) := \{a_\mathcal{Y} \in \Gamma_\mathcal{Y} \mid (s, a_\mathcal{X}, a_\mathcal{Y}) \models \rho_s\}$,

An execution (or play) between the uncontrolled environment and controlled system corresponds to a path on the game graph, beginning at a state satisfying the initial conditions θ_e and θ_s. A strategy on a game graph specifies for each path in the system and an environment move, a system move enabled at that point. We say that a strategy is winning for a GR(1) formula, if every path which conforms to the strategy satisfies all system goals or blocks at least one environment goal. It is a well-known result that for GR(1) specifications, there exists a finite memory winning strategy. Hence, we define a strategy automaton.

A *strategy automaton* for a GR(1) formula φ is a triple $A = (V, \delta, L)$, such that V is a set of nodes, $L : V \to \Gamma$, and $\delta : (\cup_{v \in V}\{v\} \times \mathrm{E_e}(L(v)) \to V$ such that for every $(v, s_\mathcal{X})$, $L(\delta(v, s_\mathcal{X})) \Downarrow_\mathcal{X} = s_\mathcal{X}$ and $L(\delta(v, s_\mathcal{X})) \Downarrow_\mathcal{Y} \in \mathrm{E_s}(L(v), s_\mathcal{X})$. We use $u \to_\delta v$ to denote $v \in \delta(u, \mathrm{E_e}(L(u)))$. Intuitively, $u \to_\delta v$ indicates there exists an environment move from state $L_1(u)$ such that automaton A transitions to node v in response.

An *execution* of a strategy automaton is a sequence $v_1 v_2 \cdots v_n$ such that $v_i \to_\delta v_{i+1}$. A *trace* of a strategy automaton is a sequence $s_1 s_2 \cdots s_n$ such that there exists and execution $v_1 v_2 \cdots v_n$ with $s_i = L(v_i)$. We say that a strategy automaton for φ is *winning* from a set of states I if $I \subseteq L(V)$, and every trace of the automaton satisfies the environmental and system goals of φ, that is, either there exists an environment goal J_j^e which is not satisfied starting from a certain point in the trace, or every system goal J_i^s is satisfied infinitely often in the trace. We simply say that a strategy automaton is winning for φ if I contains the set of states satisfying θ_e and θ_s.

B. GR(1) synthesis algorithm

We present a method for synthesizing a strategy automaton for a GR(1) formula. We first define a predecessor operator. For a given set of states $X \subseteq \Gamma$, the predecessor operator Pre returns all states from which, given any possible environment move, there is some system move going to X. The operation is a well known primitive in most algorithms for reachability computations on game graphs or in other control problems with disturbances. It is also sometimes written as $\textcircled{\Diamond}$. Precisely, let $X \subseteq \Gamma$. Define

$$\mathrm{Pre}_\varphi(X) := \big\{s \in \Gamma \mid \forall a_\mathcal{X} \in \mathrm{E_e}(s), \exists a_\mathcal{Y} \in \mathrm{E_s}(s, a_\mathcal{X}) : (a_\mathcal{X}, a_\mathcal{Y}) \in X\big\}. \quad (2)$$

We now sketch one method to synthesize a strategy for a GR(1) specification, e.g., as described in [13]. The present paper only addresses strategies constructed in this manner. Considering only such strategies is without loss of generality because a realizable specification always admits this synthesis method. For each goal J_i^s, $i \in \{0, \ldots, n-1\}$, the basic approach during synthesis is to find at each step, a move taking the game strictly closer to a J_i^s-state (a state satisfying J_i^s) or to move in such a way that one of the environment liveness conditions J_j^e, $j \in \{0, \ldots, m-1\}$, is blocked. To be more precise, consider a row of the μ-calculus formula presented in [12] to compute the winning set W_φ for a specification φ,

$$\mu Y \left(\bigvee_{j=0}^{m-1} \nu X \big((J_i^s \wedge \mathrm{Pre}_\varphi(Z_{i+1})) \vee \mathrm{Pre}_\varphi(Y) \vee \big(\neg J_j^e \wedge \mathrm{Pre}_\varphi(X)\big)\big) \right) \quad (3)$$

where subscript addition is modulo n. For fixed values of Z_i's, the μY operators are used to compute the sets which reach corresponding systems goals. Let us denote the Y sets obtained in the consecutive iterations for goal i by $Y_i^0 Y_i^1 \cdots$. These sets have the property that $Y_i^0 \subseteq Y_i^1 \subseteq \cdots Y_i^k$, and due to the finiteness of the state space, $Y_i^k = Y_i^{k+1}$ for some k. Let k be the minimal integer for which $Y_i^k = Y_i^{k+1}$. Then, it also ensures that there is a strategy that "progresses" towards the goal. For every state in Y_i^k, $k > 1$, the strategy ensures that for any environment move, there is a system move which results in a state $Y_i^{k'}$ for some $k' < k$, or remains in some state Y_i^k, but violates one of the environment goals, that is the current and next state both violate a particular environment goal. Finally, when the Z_i's reach the fixed points (cf. [12]), every system goal that does not eventually globally violate an environment goal has a strategy to move to a state in some Y_{i+1}^k.

In order to propose a patching algorithm, we need to assume that the initial strategy automaton corresponding to the unmodified GR(1) formula has more information than as defined previously. We use the above observations of the synthesis algorithm to propose a data structure on the strategy automaton, which gives an alternate characterization for the existence of a winning strategy for the GR(1) formula.

Definition 1: A *reach annotation* on a strategy automaton $A = (V, \delta, L)$ for a GR(1) formula φ is a function $\mathrm{RA} : V \to [n] \times \mathbb{Z}_+$ which satisfies the following conditions. Define RA_i for $i = 1, 2$ as $\mathrm{RA}_i(v) = n_i$, where $\mathrm{RA}(v) = (n_1, n_2)$, and let $[n] = \{0, \cdots, n-1\}$. Given $i < j$, the numbers between i and j are $i+1, \cdots, j-1$, and if $j \leq i$, then the numbers between i and j are $i+1, \cdots, n-1, 0, \cdots, j-1$.

1) $\mathrm{RA}_2(v) = 0$ iff $L(v)$ is a $\mathrm{RA}_1(v)$-system goal.
2) For each $v \in V$ and $s_{\mathcal{X}} \in \mathrm{E_e}(L(v))$, if $\mathrm{RA}_2(v) \neq 0$, then $\mathrm{RA}_1(v) = \mathrm{RA}_1(\delta(v, s_{\mathcal{X}}))$ and either
   a) there exists an environment goal J_j^e such that both $L(v)$ and $\delta(v, s_{\mathcal{X}})$ do not satisfy J_j^e, or
   b) $\mathrm{RA}_2(v) > \mathrm{RA}_2(\delta(v, s_{\mathcal{X}}))$.
3) For each $v \in V$ and $s_{\mathcal{X}} \in \mathrm{E_e}(L(v))$, if $\mathrm{RA}_2(v) = 0$, then there exists a j such that for all k between $\mathrm{RA}_1(v)$ and j, $L(v)$ is k-system goal, and $\mathrm{RA}_1(\delta(v, s_{\mathcal{X}})) = j$.

Remark 2: Note that one can easily extract a strategy automaton and a reach annotation from the Y_i^j sets defined in the strategy synthesis algorithm above.

The next theorem provides a characterization of a winning strategy in terms of the reach annotation.

Theorem 3: There exists a winning strategy automaton A for a GR(1) formula φ if and only if there exists a strategy automaton with all initial states for φ and with a reach annotation for A.

Proof: The "only if" part follows from the above remark. Conversely, let A be a strategy automaton and RA be a reach annotation on A. Let s_0 be an initial state for φ. By hypothesis there is a node v_0 in A with $L(v_0) = s_0$. Let $v_0 v_1 v_2 \cdots$ be an infinite execution originating at v_0. Note that by definition of strategy automata, such an infinite execution will occur if the environment moves are always possible (i.e., in $\mathrm{E_e}(L(v_i))$ for each v_i). Otherwise, halting on a finite execution means that the environment has violated the assumption part of the GR(1) formula ψ and therefore the resulting trace is winning. There are three possibilities for the execution $v_0 v_1 v_2 \cdots$.

1) First, there is some k such that $\mathrm{RA}_2(v_i) = \mathrm{RA}_2(v_{i+1}) \neq 0$ for $i \geq k$. Then by definition of reach annotation one of the environment goals is indefinitely violated and therefore the corresponding trace is winning.
2) Second, there is some k such that $\mathrm{RA}_2(v_i) = \mathrm{RA}_2(v_{i+1}) = 0$ for $i \geq k$. Then all system goals are satisfied at each step of the trace after k, and therefore the trace is winning.
3) Finally, there are infinitely many i where $\mathrm{RA}_2(v_i) \neq \mathrm{RA}_2(v_{i+1})$. If there exists k such that $\mathrm{RA}_2(v_i) \leq \mathrm{RA}_2(v_{i+1})$ for $i \geq k$, then because there are infinitely many indices where this inequality is strict, it must be that an environment goal is violated at v_k onward and therefore the corresponding trace is winning. Otherwise, there are infinitely many i where $\mathrm{RA}_2(v_i) > \mathrm{RA}_2(v_{i+1})$. Because $\mathrm{RA}_2(V)$ is bounded below by 0, it follows that there are infinitely many indices i where $\mathrm{RA}_2(v_i) = 0$. Then, the third property of reach annotation ensures that all system goals are visited, and therefore the corresponding trace is winning.

■

III. Patching Algorithm

In this section, we propose an algorithm for patching a strategy or controller when there is a change in the edge set associated with a game graph. Let us fix a GR(1) formula φ and a strategy automaton $A = (V, \delta, L)$ for φ with a reach annotation RA for the rest of the section.

A. Game changes

Suppose that the edges of the game graph associated with the GR(1) formula φ change from $\mathrm{E_e}$ and $\mathrm{E_s}$ to $\mathrm{E_e}'$ and $\mathrm{E_s}'$, respectively. This affects a winning strategy automaton A for φ in two ways:

1) (removal of outgoing edges from controlled vertices) Some control decisions can become unavailable. Let v be a node in A affected in this manner. Then:
$$\mathrm{Cond}_1(v) := \exists s_{\mathcal{X}} : L\left(\delta(v, s_{\mathcal{X}})\right) \Downarrow_{\mathcal{Y}} \notin \mathrm{E_s}'(L(v), s_{\mathcal{X}}). \tag{4}$$
2) (addition of outgoing edges to uncontrolled vertices) The environment has new moves. A node v is affected in this manner if:
$$\mathrm{Cond}_2(v) := \mathrm{E_e}'(L(v)) \setminus \mathrm{E_e}(L(v)) \neq \emptyset. \tag{5}$$

The set of all nodes v satisfying Cond_1 or Cond_2 are said to be the *affected* nodes of A due to the changes $\mathrm{E_e}'$ and $\mathrm{E_s}'$ to the edges $\mathrm{E_e}$ and $\mathrm{E_s}$ of the game graph of φ.

B. Overview

Consider the case where parts of the specification abstract the behavior of a dynamical control system. The precise meaning of "abstraction" is beyond the scope of this paper; we refer the reader to [7] and references therein for details. Suffice it to observe that, while the abstraction contributes variables (i.e., elements of $\mathcal{X} \cup \mathcal{Y}$) and transition rules (i.e., conjuncts in ρ_e or ρ_s; recall that for state formulae f and g, we have that $\Box(f \wedge g) \equiv (\Box f) \wedge (\Box g)$) in a transparent manner, the underlying dynamics provide additional structure we hope to exploit. To that end, we assume we can compute a set of vertices in the game graph $(\Gamma, \mathrm{E_e}, \mathrm{E_s})$ that is "local" in some sense. For instance, this could be obtained by computing the norm ball centered at some state of the underlying dynamical system and finding all vertices in Γ corresponding to states contained in that norm ball [10]. We call this set $\hat{N}$ (recall $\hat{N} \subseteq \Gamma$) and refer to it as the "neighborhood of vertices." This terminology is motivated by how we *construct* $\hat{N}$ rather than a topological space defined for the game graph $(\Gamma, \mathrm{E_e}, \mathrm{E_s})$. The precise conditions under which $\hat{N}$ possesses desirable properties, such as being a small subset of Γ or preserving local connectivity of the graph $(\Gamma, \mathrm{E_e}, \mathrm{E_s})$, is the subject of future work.

Note that the reach annotation RA defines a partition of the nodes in A such that the nodes with $\mathrm{RA}_1(v) = i$ correspond to those reaching system goal i or eventually

globally violating an environment goal. Upon reaching a state satisfying J_i^s, the goal mode is incremented, modulo n, and the process continues. (Note that a state may satisfy more than one system goal.) The broad idea of the patching algorithm is to identify the affected nodes corresponding to each system goal and locally modify the automaton.

Suppose that the set of affected nodes due to the edge changes is non-empty. Let us denote the modified formula as φ', that is, the formula is the same as φ, except that the ρ_e and ρ_s are replaced by ρ_e' and ρ_s' corresponding to edge changes in E_e and E_s. For each goal mode i, let U_i be the set of nodes in the strategy automaton affected by the change, which are not themselves labeled with a goal state. More precisely,

$$U_i := \big\{ v \in V \mid L(v) \not\models J_i^s \wedge (\mathrm{RA}_1(v) = i) \\ \wedge (\mathrm{Cond}_1(v) \vee \mathrm{Cond}_2(v)) \big\}. \quad (6)$$

Observe that these sets are disjoint, and that one or more of them may be empty. Recall that nodes labeled with the same state have different goal modes. This is assumed without loss of generality because otherwise the strategy is redundant (not minimal).

Next, we define an operator for producing local strategies. Let $\hat{N}$ and $B \subseteq \Gamma$ be sets of states. Define a μ-calculus formula

$$\mathrm{Local}(B, \hat{N}) := \mu Y \Bigg(\bigvee_{j=0}^{m-1} \nu X \Big(B \vee \Big(\mathrm{Pre}_{\varphi'}(Y) \wedge \hat{N} \Big) \\ \vee \Big(\neg J_j^e \wedge \mathrm{Pre}_{\varphi'}(X) \wedge \hat{N} \Big) \Big) \Bigg). \quad (7)$$

This formula can be viewed as a truncated form of that presented in [12] (also see equation (3)) to compute the winning set for a GR(1) specification. Notice that states returned by the Pre operation are restricted to $\hat{N}$. Let us denote the set of states in the fixed point computation of $\mathrm{Local}(B, \hat{N})$ as $[\![\mathrm{Local}(B, \hat{N})]\!]$. The result is a strategy which ensures that either some state in B is eventually reached or some environment goal is eventually persistently violated. Let us denote by (A', S_1, S_2) the fact that A' is a strategy automaton for φ', which is winning from any state in S_1 with respect to reaching some state in S_2, or eventually globally violating an environmental goal of φ' (same as that of φ). Note that there exists a A' such that $(A', [\![\mathrm{Local}(B, \hat{N})]\!], B)$. It can be computed using the fixed point algorithm defined by the formula $\mathrm{Local}(B, \hat{N})$.

Further, we can ensure the existence of the following partial reach annotation.

Theorem 4: Given (A', C, B), where $A' = (V, \delta, L)$ and $C \subseteq [\![\mathrm{Local}(B, \hat{N})]\!]$, there exists a *partial reach annotation* $\mathrm{RA} : V \to \{i\} \times \mathbb{Z}_+$, for any i such that the following hold:

1) $\mathrm{RA}_2(v) = 0$ iff $L(v)$ is in B.
2) For each $v \in V$ and $s_{\mathcal{X}} \in \mathrm{E_e}(v)$, if $\mathrm{RA}_2(v) \neq 0$, then either
   a) there exists an environment goal J_j^e such that both $L(v)$ and $\delta(v, s_{\mathcal{X}})$ do not satisfy J_j^e, or
   b) $\mathrm{RA}_2(v) > \mathrm{RA}_2(\delta(v, s_{\mathcal{X}}))$.

Proof: This is immediate from the definition of the μ-calculus formula 7 and using the same construction from intermediate values of the fixed point computation of $[\![\mathrm{Local}(B, \hat{N})]\!]$, as in Remark 2. ∎

Remark 5: It is obvious from the proof that the above theorem is constructive, that is, one can compute a partial reach annotation for any $(A', [\![\mathrm{Local}(B, \hat{N})]\!], B)$.

We assume at least one of the states in the automaton satisfying each goal remains feasible, i.e., there is some way to drive the play to it after the edge set change. In other words, we assume that we do not need to introduce new goal nodes into the automaton A. By construction of the strategy, the goal mode must change when a play leads to one of these nodes. Under this assumption, it suffices to adjust *how* we reach a goal state, but once there the play will be at an existing node of A.

In summary, the main steps of the proposed method are as follows. First compute a neighborhood $\hat{N} \subset \Gamma$ of states based on proximity in the underlying continuous space, such that $\hat{N}$ is sufficiently large to properly contain all states of nodes in $\bigcup_i U_i$. See discussion in Section IV about possibilities for choosing $\hat{N}$. For each goal mode i with nonempty U_i,

1) Create the set N^i of nodes with goal mode i and corresponding state in $\hat{N}$.
2) Create the set ***Entry***^{i} of labels of nodes in N^i that can be entered from outside N^i in the original strategy. Also include in this set the current state of the play (assuming the algorithm is being used online, we must address how to move from the current position).
3) Compute the minimum $\mathrm{RA}_2(v)$ value (related to the reach annotation yielded by the automaton) over all nodes in $\mathrm{RA}^{-1}(\{i\} \times$ ***Entry***$^i) \cup U_i$, and call it m^*.
4) Create the set ***Exit***^{i} of states of nodes v in N^i with $\mathrm{RA}_2(v)$ less than m^*. Precisely,
$$\textit{Exit}^i := \big\{ L(v) \,|\, v \in N^i, \mathrm{RA}_2(v) < m^* \big\}.$$
5) Compute $[\![\mathrm{Local}($***Exit***$^i, \hat{N})]\!]$.
   - If ***Entry***^{i} is not contained in $[\![\mathrm{Local}($***Exit***$^i, \hat{N})]\!]$, then the local problem is declared unrealizable and the neighborhood $\hat{N}$ must be adjusted and the above steps repeated.
   - Otherwise, compute the local strategy $(A^i,$ ***Entry***$^i,$ ***Exit***$^i)$, and use it to mend the original strategy A by removing nodes in N^i and replacing it by the nodes in A^i, and adding appropriate edges corresponding to the edges into the entry states and the edges from the exit nodes in A.

Intuitively, the sets are constructed such that if we can get to *Exit* from each node in *Entry*, then from the definition of reach annotation it must be that distance from the current target system goal has not increased and that there is not a path under the new automaton A' leading back to *Entry*.

To improve efficiency, Pre operations in the iterations are restricted to the neighborhood $\hat{N}$. Note that upon reaching a "border node," we must have arrived at a state in *Entry* (by definition) and therefore restricting the Pre operation to not go outside $\hat{N}$ does not constrain the method.

When synthesizing for each U_i, incorporate the reach annotation from the patch into the original automaton A by scaling all L_3 values of A such that there is sufficient "room" for the new range of reach annotation values from the patch, after offsetting by the value at the lower attachment point. The resulting automaton A' then has a labeling L'_3 that yields a reach annotation, as shown in Section III-D. Thus the proposed method can be applied to A', the output of which can again be patched, and so on.

C. Formal statement

A precise statement appears in Algorithms 1 and 2. Some clarifications follow.

- $\hat{N} \subset \Gamma$ is a precondition asserting a set of neighborhood states is given. Note that $\hat{N}$ may be defined by a predicate on $\mathcal{X} \cup \mathcal{Y}$.
- Line 20 finds the minimum positive integer that ensures the distance between the *Entry* and *Exit* sets (given by $m^* - m_*$) is greater than the range of reach annotation values in the local strategy (given by m_{local}).

Algorithm 1 Find local strategies

1: INPUT: GR(1) formula φ, strategy A, reach annotation RA, modified formula φ', neighborhood $\hat{N} \subset \Gamma$
2: OUTPUT: set of triples $(A^i, \mathit{Entry}^i, \mathit{Exit}^i)$ with partial reach annotation RA^i
3: `Patches` $:= \emptyset$
4: **for all** i such that $U_i \neq \emptyset$ **do**
5: $\quad N^i := \{v \in V \mid L(v) \in \hat{N}, \mathrm{RA}_1(v) = i\}$
6: $\quad$ **if** $U_i \setminus N^i \neq \emptyset$ **then**
7: $\quad\quad$ error — N^i is too small.
8: $\quad$ **end if**
9: $\quad \mathit{Entry}^i := \{L(v) \mid v \in N^i, \exists u \in V \setminus N^i : u \to_\delta v\}$
10: $\quad m^* := \min_{v \in U_i \cup \left(L^{-1}(\mathit{Entry}^i) \cap \mathrm{RA}_1^{-1}(i)\right)} \mathrm{RA}_2(v)$
11: $\quad \mathit{Exit}^i := \{L(v) \mid v \in N^i, \mathrm{RA}_2(v) < m^*\}$
12: $\quad Z := [\![\mathrm{Local}(\mathit{Exit}^i, \hat{N})]\!]$
13: $\quad$ **if** $\mathit{Entry}^i \not\subseteq Z$ **then**
14: $\quad\quad$ error — local problem unrealizable.
15: $\quad$ **else**
16: $\quad\quad$ Synthesize A^i such that $(A^i, \mathit{Entry}^i, \mathit{Exit}^i)$ with partial reach annotation RA^i
17: $\quad\quad$ `Patches` $:=$ `Patches` $\cup \{((A^i, \mathit{Entry}^i, \mathit{Exit}^i), N^i, \mathrm{RA}^i)\}$
18: $\quad$ **end if**
19: **end for**
20: **return** `Patches`

D. Analysis

We begin with some remarks about the algorithm.

Remark 6: For each $i \in \{0, \ldots, n-1\}$, Exit^i (see definition on Line 11 of Algorithm 1) does not share any states with Entry^i or $L(U_i)$. Precisely,

$$\mathit{Exit}^i \cap \left(\mathit{Entry}^i \cup L(U_i)\right) = \emptyset.$$

As described in Section II-A, the safety formulae ρ_e and ρ_s of a GR(1) specification can be equivalently expressed by a game graph $(\Gamma, \mathrm{E_e}, \mathrm{E_s})$. In our analysis below we consider changes to ρ_e and ρ_s, which is equivalent to the removal or addition of edges in E, denoted by $E \triangle E' \neq \emptyset$ (where E' is the new edge set). The other parts of the specification remain unchanged.

Algorithm 2 Merge local strategies into the original

1: INPUT: GR(1) formula φ, strategy A, reach annotation RA, modified formula φ', neighborhood $\hat{N} \subset \Gamma$, and Patches from Algorithm 1.
2: OUTPUT: Strategy automaton A' for φ' with reach annotation RA'
3: **for all** $\left((A^i = (V^i, \delta^i, L^i), \mathit{Entry}^i, \mathit{Exit}^i), N^i, \mathrm{RA}^i\right) \in$ `Patches` **do**
4: $\quad$ **for all** $s \in \mathit{Entry}^i$ **do**
5: $\quad\quad$ find $u \in V^i$ such that $L^i(u) = s$
6: $\quad\quad$ **for all** $v \in V \setminus N^i$ such that $v \to_\delta u'$ for some $u' \in N^i$ with $L(u') = s$ **do**
7: $\quad\quad\quad$ replace transition $v \to_\delta u'$ of δ by $v \to_\delta u$
8: $\quad\quad$ **end for**
9: $\quad$ **end for**
10: $\quad m_* := \max_{v \in L^{-1}(\mathit{Exit}^i) \cap \mathrm{RA}_1^{-1}(i)} \mathrm{RA}_2(v)$
11: $\quad$ **for all** $s \in \mathit{Exit}^i$ **do**
12: $\quad\quad$ **for all** $u \in L^{-1}(\mathit{Exit}^i) \cap \mathrm{RA}_1^{-1}(i)$ with $u \to_\delta v$ **do**
13: $\quad\quad\quad$ append transition $u' \to_\delta v$ for every $u' \in N^i$ with $L^i(u') = s$
14: $\quad\quad$ **end for**
15: $\quad$ **end for**
16: $\quad m_{\mathit{local}} = \max_{v \in V^i} \mathrm{RA}_2^i(v)$
17: $\quad$ **for all** $v \in V^i$ **do**
18: $\quad\quad \mathrm{RA}_2^i(v) := \mathrm{RA}_2^i(v) + m_*$
19: $\quad$ **end for**
20: $\quad \alpha := \min_{k \in \mathbb{Z}_+, k(m^* - m_*) > m_{\mathit{local}}} k$
21: $\quad$ **for all** $v \in V$ with $\mathrm{RA}_1(v) = i$ **do**
22: $\quad\quad \mathrm{RA}_2(v) := \alpha \cdot \mathrm{RA}_2(v)$
23: $\quad$ **end for**
24: $\quad$ Delete all N^i //Recall that $N^i \subset V$
25: $\quad V := (V \setminus N^i) \uplus V^i$
26: $\quad$ Remove edges from δ which intersect with N^i
27: $\quad$ Add all edges from δ^i to δ
28: $\quad$ Extend L to include nodes in V^i
29: $\quad A' := (V, \delta, L)$ and $\mathrm{RA}' := \mathrm{RA}$.
30: **end for**

Let φ be a GR(1) specification as in equation (1). Denote the corresponding game graph by $(\Gamma, \mathrm{E_e}, \mathrm{E_s})$. Suppose that the safety formulae ρ_e and ρ_s of φ are changed, and call the modified specification φ' with safety formulae ρ'_e and ρ'_s. Let $A = (V, \delta, L)$ denote the original strategy automaton whose L yields a reach annotation, and suppose $N \subset \Gamma$ is chosen such that the patching problem is feasible, i.e., the algorithm terminates producing a modified automaton $A' = (V', \delta', L')$.

Theorem 7: The output of Algorithm 2, namely, A' and RA' is such that A' is a strategy automaton for φ' and RA' is a reach annotation on A' with respect to φ'. Hence, A' is a winning strategy automaton for φ'.

Proof: By Theorem 3, it suffices to show that A' is a strategy automaton, and that RA' is a reach annotation for it. For each system goal mode i, on lines 24–29 of Algorithm 2 all nodes labeled with states affected by the GR(1) formula change to φ' are replaced. Thus, all nodes in A' have an outgoing edge for each possible environment move, leading to a permissible system move. Therefore A' is a strategy automaton.

By hypothesis, the original strategy automaton A has a reach annotation RA and each patch automaton A^i such that $(A^i, \mathit{Entry}^i, \mathit{Exit}^i)$ has partial reach annotation RA^i. If describing a strategy automaton, a partial reach annotation may be regarded as a reach annotation with constant RA^i_1. From these observations, it suffices to consider the transitions from A into A^i, and A^i into A. Lines 10–23 of Algorithm 2 ensure that RA_2 is nonincreasing across these transitions. It follows that RA' is a reach annotation on A'. ∎

IV. Example scenarios and experiments

The selection of the neighborhood $\hat{N}$ plays a crucial role in our proposed method, but may not admit a general form. Instead, one may need to have templates for constructing $\hat{N}$ for various problem classes and hybrid control systems. Below we consider discovery of a static obstacle online. Some other possibilities include the following.

- $\hat{N}$ could be a norm ball centered at some point of interest and scaled as needed. This is the heuristic used in [10], where the radius is incremented until all local problems are realizable, or the global problem is recovered.
- $\hat{N}$ could be obtained from the short-time reachable space computed for the given robot dynamics. This could be useful for under-actuated systems.
- If a nominal plan is obtained using a gradient-based method as in [14], then $\hat{N}$ may be iteratively expanded toward a goal, while allowing sufficient "width" for reactivity in an unknown environment.
- To find an appropriate neighborhood $\hat{N}$ after obtaining a finer resolution cell decomposition, include all strategy automaton nodes with state corresponding to refined cells. Depending on dynamics and continuity of the underlying system, we also include immediately neighboring cells in $\hat{N}$. Selecting which unrefined cells to include appears easier in some cases, such as for linear systems, and is the subject of future work.

A. Unreachable cells

One of the basic approaches to robotic navigation is decomposition of the workspace (e.g., an office floor) into finitely many cells (for an overview, see [15] and [6]). If we can assure the existence of dynamical control laws for steering the robot among these cells, then the cell decomposition may be representable as a finite directed graph, and hence planning over the workspace becomes a reactive synthesis problem. To cope with uncertainty of the space being navigated, a robot may build and update a map as new sensor data is collected online. An important class of such changes is the case where a cell becomes unnavigable. For instance, there may be a static obstacle. In this scenario, the strategy must be adjusted to move around the occupied cell. If the obstacle is small relative to the overall space addressed by the strategy, then we may expect the proposed method to be advantageous.

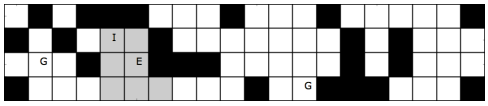

Fig. 1. 4×20 gridworld. As a graph, it is 4-connected, i.e., moves can only be "up," "down," "left," or "right." **I** indicates the initial robot position, **G** indicate goal cells. The gray cells can be occupied by a dynamic obstacle, which must repeatedly visit the **E** cell (environment liveness condition).

TABLE II

Ratio of time to run our patching algorithm to global re-synthesis for random "unreachable cell" gridworld problems.

block density	N	min	mean	max	std dev
.1	100	0.0047	0.1902	0.5314	0.0952
.3	100	0.1391	0.4039	0.7424	0.0950
.5	98	0.2581	0.5561	0.8940	0.0985
.7	108	0.3439	0.6309	1.3276	0.1696

As preliminary evaluation of the proposed method, we conducted a simulation experiment for this scenario in a small gridworld problem. An illustration of the setting is given in Figure 1. Random gridworlds were generated for block densities of 0.1, 0.3, 0.5, and 0.7. E.g., a block density of 0.1 means that 10% of cells are occupied. At each time step, the robot can move "up," "down," "left," or "right" (i.e., the world is 4-connected). There are two goals and one initial position, all randomly placed (but not overlapping). A single dynamic obstacle is also present. It must always eventually return to its base cell (labeled **E** in Figure 1), but on any particular instant may be at most one step away from its base (gray cells in Figure 1). The dynamic obstacles base cell is chosen randomly, and there may exist a plan for the robot to avoid it entirely. For each trial, a gridworld problem is randomly generated, solved to obtain a nominal strategy, and then a new static obstacle is introduced. The location of the new block is random but restricted to guarantee interference with the nominal strategy. For 0.7 density trials, the world has size 6×20; for all others, it has size 4×20.

The neighborhood $\hat{N}$ is formed by a 3×3 subgraph centered on the newly blocked cell. Times required for global re-synthesis and times for our proposed method are shown in Figure 2, where two substantial outliers were excluded for clarity. Outliers were $(319.25, 1.51)$ and $(88.02, 2.18)$, where (global, patching) times are in seconds. Several statistics are given in Table II. Figure 3 shows mean run-times for global re-synthesis and the proposed method with respect to gridworld block density.

V. Experimental Demonstration

In addition to simulations, the patching algorithm was demonstrated on a physical setup consisting of a tracked

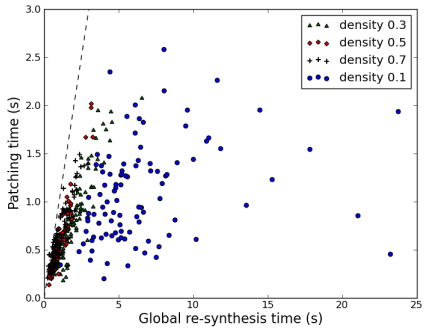

Fig. 2. Simulation run-times of global re-synthesis and our proposed method after introduction of a static obstacle. Slope 1 (unity ratio) is indicated by a dashed line. Does not include two outliers: $(319.25, 1.51)$ and $(88.02, 2.18)$, where (global, patching) times are in seconds.

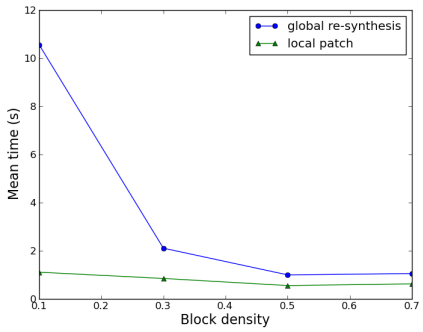

Fig. 3. Mean run-times of global re-synthesis and our proposed method after introduction of a static obstacle. Notice that for increasing densities, global re-synthesis takes less time, as may be expected given fewer reachable cells in the gridworld.

robot navigating through a planar environment with unexpected obstacles.

The environment through which the robot navigates is a $3\text{m} \times 3\text{m}$ tiled floor populated with stationary obstacles. Each goal J_i^s is inserted in software, and an automaton is synthesized to move between them. Patches to the strategy automaton are made as new obstacle position data become available online.

The robot used in these experiments was an iRobot LANdroid (Figure 4). The LANdroid is equipped with a Hokuyo scanning range finder (URG-04LX-UG01) and three IR LEDs, used in conjunction with an overhead position tracking system based on FView [16] to determine the position and heading of the LANdroid as it navigates through its environment. ROS [17] nodes were used to receive data from the Hokuyo and send control commands to the LANdroid, as well as to receive position information; the majority of processing was done off-robot. Plans and patches were synthesized using TuLiP[1] and gr1c[2].

The discrete abstraction used in this hardware setup took the form of a quadtree data structure—refer to Figure 5 for an example. Nodes of the quadtree correspond to regions

[1]http://tulip-control.sf.net
[2]http://scottman.net/2012/gr1c

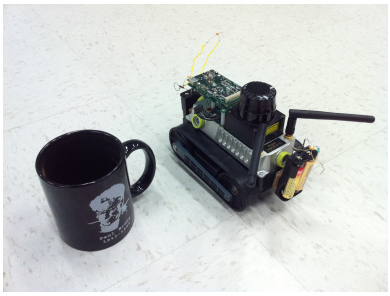

Fig. 4. iRobot LANdroid platform used in experiments. Hokuyo scanning range finder and IR LEDs (with battery packs) can be seen.

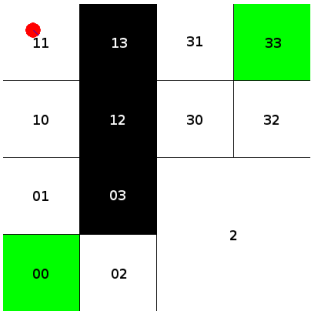

Fig. 5. A rendering of a quadtree as used in the experimental implementation. Each cell is labeled with its respective tesseral address.

in a planar environment, and refinement and coarsification of quadtree nodes represent the splitting and joining of regions to and from four interior subregions. The level of refinement in a given quadtree cell corresponds to occupancy information retrieved from a probabilistic occupancy grid map populated by data from the laser range finder. The intermediate occupancy grid layer helps avoid unnecessary refinement of the quadtree due to noise.

The quadtree utilizes tesseral addressing [18] to allow for rapid adjacency calculations. Tesseral addresses are well-suited to representing hierarchical tessellations of planar spaces, as the quadtree in this implementation is doing. Here, tesseral addresses take the form of quaternary strings which uniquely address a region in a 2-D plane. For example, the tesseral address "30" would be read as "the 0th subregion of the 3rd subregion of the plane". A labeled quadtree can be seen in Figure 5. In this implementation, the neighborhood $\hat{N}$ was determined based on cell adjacency and ancestry in the quadtree.

The process of patching in the context of these experiments can be seen in Figure 6. A video demonstration is available at `http://vimeo.com/49653485`. Initially, a path visiting two goals (green cells) is being pursued over the quadtree according to a nominal strategy automaton. As new occupancy data becomes available online, the automaton must be patched in response. $\hat{N}$ can be seen in red (the refined quadtree cell) and orange (non-refined neighbors); the partial resynthesis is made using cells within this neighborhood, and a correct strategy is recovered.

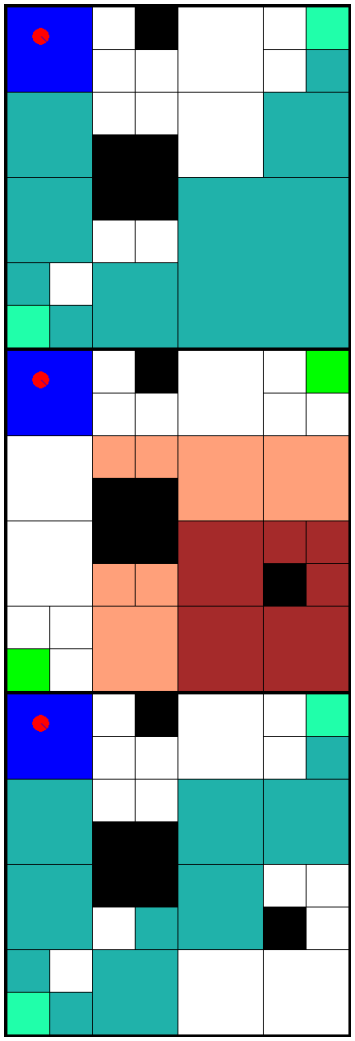

Fig. 6. Path from a plan before (top), during (middle), and after patching.

VI. Conclusion and future work

It follows from Theorem 7 that given an appropriate initial strategy, we may patch-and-repeat indefinitely while maintaining correctness. However a critical issue in this process is selection of $\hat{N}$ and realizability of each successive specification. While these are informally explored in the computational experiments presented in Section IV, a thorough investigation remains for future work.

Current laboratory work described in Section V has focused on demonstrating viability and establishing a hardware setup to be used for future experiments. Future work will include statistical comparison of time required when patching and when fully resynthesizing. Additionally we plan to include nondeterministic elements in the robots environment, such as a moving obstacle.

The key insight of the present work is twofold. Let A be the original strategy automaton. First, by finding the set U_i of nodes in A that are affected for each goal J_i^s, we exploit the case of problems where system goals correspond to disconnected regions of a continuous space. In this case, only efforts to reach some of the goals may be affected by a game graph change. For instance, this naturally occurs in robot surveillance problems, where different rooms in a building must be visited. Second, our local μ-calculus formula (7) has alternation depth of one (cf. [19]), and indeed, given environment goals, is just a reachability computation around a cluster of affected vertices in the game graph. In future work we will investigate the computational complexity of the proposed method and conduct empirical studies in application to robot navigation.

Acknowledgments

This work is partially supported by the Boeing Corporation, the Caltech Center for the Mathematics of Information (CMI), and a United Technologies Research Center postdoctoral fellowship.

References

[1] J. M. Davoren and A. Nerode, "Logics for hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 985–1010, July 2000.

[2] J. G. Thistle and W. M. Wonham, "Control problems in a temporal logic framework," *International Journal of Control*, vol. 44, no. 4, pp. 943–976, 1986.

[3] M. S. Branicky, "Universal computation and other capabilities of hybrid and continuous dynamical systems," *Theoretical Computer Science*, vol. 138, pp. 67–100, 1995.

[4] G. Lafferriere, G. J. Pappas, and S. Sastry, "O-minimal hybrid systems," *Mathematics of Control, Signals, and Systems*, vol. 13, pp. 1–21, 2000.

[5] R. Majumdar, E. Render, and P. Tabuada, "Robust discrete synthesis against unspecified disturbances," in *Hybrid Systems: Computation and Control (HSCC)*, April 2011.

[6] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: http://planning.cs.uiuc.edu/

[7] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, July 2000.

[8] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[9] T. Wongpiromsarn, "Formal methods for design and verification of embedded control systems: Application to an autonomous vehicle," Ph.D. dissertation, California Institute of Technology, 2010.

[10] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, Minnesota, USA, May 2012, pp. 5163–5170.

[11] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[12] Y. Kesten, N. Piterman, and A. Pnueli, "Bridging the gap between fair simulation and trace inclusion," *Information and Computation*, vol. 200, pp. 35–61, 2005.

[13] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *In Proc. 7th International Conference on Verification, Model Checking and Abstract Interpretation*, ser. Lecture Notes in Computer Science, vol. 3855. Springer, 2006, pp. 364–380. [Online]. Available: http://jtlv.sourceforge.net/

[14] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 5, pp. 501–518, October 1992.

[15] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2006.

[16] A. Straw and M. Dickinson, "Motmot, an open-source toolkit for realtime video acquisition and analysis," *Source Code for Biology and Medicine*, vol. 4, no. 1, p. 5, 2009.

[17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2, 2009.

[18] F. Coenen. (2000, Oct.) Tesseral addressing. [Online]. Available: http://www.csc.liv.ac.uk/∼frans/OldResearch/dGKBIS/tesseral.html

[19] E. A. Emerson, C. S. Jutla, and A. P. Sistla, "On model checking for the μ-calculus and its fragments," *Theoretical Computer Science*, vol. 258, pp. 491–522, 2001.